

Resumptions, Weak Bisimilarity and Big-Step Semantics for While with Interactive I/O: An Exercise in Mixed Induction-Coinduction

Keiko Nakata and Tarmo Uustalu

Institute of Cybernetics at Tallinn University of Technology, Akadeemia tee 21, EE-12618 Tallinn, Estonia

{keiko|tarmo}@cs.ioc.ee

We look at the operational semantics of languages with interactive I/O through the glasses of constructive type theory. Following on from our earlier work on coinductive trace-based semantics for While [17], we define several big-step semantics for While with interactive I/O, based on resumptions and termination-sensitive weak bisimilarity. These require nesting inductive definitions in coinductive definitions, which is interesting both mathematically and from the point-of-view of implementation in a proof assistant.

After first defining a basic semantics of statements in terms of resumptions with explicit internal actions (delays), we introduce a semantics in terms of delay-free resumptions that essentially removes finite sequences of delays on the fly from those resumptions that are responsive. Finally, we also look at a semantics in terms of delay-free resumptions supplemented with a silent divergence option. This semantics hinges on decisions between convergence and divergence and is only equivalent to the basic one classically. We have fully formalized our development in Coq.

1 Introduction

Interactive programs are those programs that take inputs, do some computation, output results, and iterate this cycle possibly infinitely. Operating systems and data base systems are typical examples. They are important programs and have attracted formal study to guarantee their correctness/safety. For instance, a web application should protect confidentiality of the data it processes in interaction with possibly untrusted agents, and a certified compiler should preserve input/output behavior of the source program in the compiled code. These works call for formal semantics of interactive programs.

Continuing our previous work [17] on a trace-based coinductive big-step semantics for potentially nonterminating programs, we present a *constructive* account of interactive input-output *resumptions*¹, their important properties, such as weak bisimilarity and responsiveness (a program always eventually performs input or output unless it terminates) and *big-step semantics* of reactive programs. We devise both constructive-style and classical-style concepts and identify their relationships. Classical-style concepts rely on upfront decisions of whether a computation is going to terminate, make an observable action, i.e., perform input or output, or silently diverge. The problem is generally undecidable. Hence, classical-style concepts tend to be too strong for constructive reasoning.

Our operational semantics are *resumption-based*. A resumption is roughly a tree representing possible runs of a program. The tree branches on inputs, each edge corresponding to each possible input, and has infinitely deep paths if the program may diverge. We begin the paper by formalizing important properties of resumptions, among which (termination-sensitive) weak bisimilarity is the most interesting

¹The word ‘resumption’ is sometimes reserved for denotations of parallel threads. We apply it more liberally to datastructures recording evolution in small steps. This usage dates back to Plotkin [20] and was reinforced by Cenciarelli and Moggi [5].

one technically, requiring nesting of induction into coinduction. We give a constructive-style formulation of weak bisimilarity and relate it to the classical-style version adapted from previous work [13, 3]. We then present three *big-step semantics* for interactive While, i.e., While extended with input/output statements: a basic semantics which explicitly deals with internal actions (delay steps) and assigns a resumption for all configurations (statement-state pairs); a delay-free semantics for responsive configurations; and a classical-style semantics, which is total classically for all configurations. The two latter semantics collapse finite sequences of delay steps on the fly. The classical-style semantics can in addition recognize silent divergence; classical-style resumptions include a distinguished element to represent divergence. Moreover, all three semantics are equivalent under suitable assumptions. Our approach with big-step semantics in terms of resumptions allow for reasoning about operational behaviors of programs in a syntax-independent way. We therefore argue that it is more abstract than approaches by means of small-step semantics, or labelled transition systems (in terms of configurations involving a residual program or a control point). To compare our big-step semantics to more traditional approaches, we also define an uncontroversial small-step semantics with an associated notion of weak bisimilarity of configurations and show that it agrees with our basic big-step semantics. These technical results form the main contributions of the paper.

Why do we want to be *constructive*? First, let us state that our choice is neither motivated nor depends on any argument of truth: we are not claiming in this paper that classical logic is less true than intuitionistic logic and none of the points we make hinge on this being the case. Nevertheless, we do think that working in a constructive logic is very useful also if one has no philosophical problem in accepting non-constructive arguments. Our reasons are these. For us, using constructive logic is primarily a technical way to be conscious about the principles we depend on in our arguments. We are by no means limiting ourselves: when we really need some non-constructive principle in a constructive argument, we can always explicitly assume this principle (or the specific instance that we need). But it so happens that a need for unexpectedly strong principles is often a sign of some imperfect design choice in the setup of a theory. Another reason to be constructive as a semanticist is that programming is about computable functions only. In constructive logic, we do not have to specifically worry about computability: only computable functions are there and can be spoken about. For example, the formula $\forall x. px \vee \neg(px)$ is not a tautology, it states that p is a decidable: there is a computable function mapping any x to a proof of px or a proof of $\neg(px)$ (so also to yes or no, should one not care about the proofs). In big-step semantics, although we specify evaluation as a relation in this paper, it is important for us that it can be turned into a function, or else we do not capture the intuitive idea that programs represent computable functions from initial configurations into behaviors.

We have *formalized* the development in Coq version 8.2pl1. This gives us greater confidence in the correctness of our reasoning, in particular regarding the productivity of coinductive proofs, since the type checker of Coq helps us avoid mistakes by ruling out unproductivity. We rely on Mendler-style coinduction to circumvent the limitations imposed by syntactic guardedness approach [9] of Coq. The Coq development is available at <http://cs.ioc.ee/~keiko/sophie.tgz>.

The language we consider is the While language extended with input and output primitives, with statements $s : stmt$ defined inductively by

$$s ::= \text{skip} \mid s_0; s_1 \mid x := e \mid \text{if } e \text{ then } s_t \text{ else } s_f \mid \text{while } e \text{ do } s_t \mid \text{input } x \mid \text{output } e$$

We assume given the sets of variables and (pure) expressions, whose elements are ranged over by metavariables x and e respectively. We assume the set of values to be the integers, non-zero integers counting as truth and zero as falsity. The metavariable v ranges over values. A state, ranged over by σ , maps variables to values. The notation $\sigma[x \mapsto v]$ denotes the update of a state σ with v at x . We assume

given an evaluation function $\llbracket e \rrbracket \sigma$, which evaluates e in the state σ . We write $\sigma \models e$ and $\sigma \not\models e$ to denote that e is true, resp. false in σ .

2 Resumptions

We will define operational semantics of interactive While in terms of states and (interactive input-output) resumptions. Informally, a resumption is a datastructure that captures all possible evolutions of a configuration (a statement-state pair), a computation tree branching according to the external non-determinism resulting from interactive input.²

Basic (delayful) *resumptions* $r : res$ are defined coinductively by the rules³

$$\frac{\sigma : state}{ret \sigma : res} \quad \frac{f : Int \rightarrow res}{in f : res} \quad \frac{v : Int \quad r : res}{out v r : res} \quad \frac{r : res}{\delta r : res}$$

so a resumption either has terminated with some final state, $ret \sigma$, takes an integer input v and evolves into a new resumption $f v$, $in f$, outputs an integer v and evolves into r , $out v r$, or performs an internal action (observable at best as a delay) and becomes r , δr . For simplicity, we assume input totality; i.e., input resumptions, represented by total functions, accept any integers. But we could instead have had them partial, e.g., by letting the constructor in take the intended domain of definedness as an additional argument. We also define (strong) *bisimilarity* of two resumptions, $r \approx r_*$, coinductively by

$$\frac{}{ret \sigma \approx ret \sigma} \quad \frac{\forall v. f v \approx f_* v}{in f \approx in f_*} \quad \frac{r \approx r_*}{out v r \approx out v r_*} \quad \frac{r \approx r_*}{\delta r \approx \delta r_*}$$

Bisimilarity is straightforwardly seen to be an equivalence. We think of bisimilar resumptions as equal, i.e., type-theoretically we treat resumptions as a setoid with bisimilarity as the equivalence relation⁴. Accordingly, we have to make sure that all functions and predicates we define on resumptions are setoid functions and predicates, i.e., insensitive to bisimilarity.

Here are some examples of resumptions, defined by corecursion:

$$\begin{aligned} \perp &= \delta \perp \\ rep \ n &= \delta (\delta (out \ n (rep \ n))) \\ rep' \ n &= \delta (out \ n (rep' \ n)) \\ echo \ \sigma &= in (\lambda n. \delta (if \ n \neq 0 \text{ then } out \ n (echo \ \sigma) \text{ else } ret \ \sigma)) \\ echo' &= in (\lambda n. \delta (if \ n \neq 0 \text{ then } out \ n \ echo' \text{ else } \perp)) \end{aligned}$$

\perp represents a resumption that silently diverges. rep outputs an integer n forever. rep' is similar but has shorter latency. Both $echo$ and $echo'$ echo input interactively; the former terminates when the input is 0, whereas the latter diverges in this situation.

Convergence, $r \downarrow r'$, states that r converges in a finite number of steps to a resumption r' , which has terminated or makes an observable action (performs input/output) as its first move. It is defined inductively by

$$\frac{}{ret \ \sigma \downarrow ret \ \sigma} \quad \frac{\forall v. f v \approx f_* v}{in f \downarrow in f_*} \quad \frac{r \approx r_*}{out v r \downarrow out v r_*} \quad \frac{r \downarrow r'}{\delta r \downarrow \delta r'}$$

²There are alternatives. We could have chosen to work, e.g., with functions from streams of input values into traces, i.e., computation paths.

³We mark inductive definitions by single horizontal rules and coinductive definitions by double horizontal rules.

⁴Classically, strong bisimilarity is equality. But we work in an intensional type theory where strong bisimilarity of colists is weaker than equality (just as equality of two functions on all arguments is weaker than equality of these two functions). E.g., \perp and $\delta \perp$ are only strongly bisimilar.

In contrast, *divergence*, $r\uparrow$, states that r diverges silently. It is defined coinductively by

$$\frac{r\uparrow}{\delta r\uparrow}$$

For instance, we have $\delta(\delta(\text{ret } \sigma)) \downarrow \text{ret } \sigma$, $\text{rep } n \downarrow \text{out } n(\text{rep } n)$ and $\perp \uparrow$.

Both convergence and divergence are setoid predicates. Constructively, it is not the case that $\forall r. (\exists r'. r \downarrow r') \vee r\uparrow$, which amounts to decidability of convergence. But classically, this dichotomy is true. In particular, $\forall r. \neg(\exists r'. r \downarrow r') \rightarrow r\uparrow$ is constructively provable, but $\forall r. \neg r\uparrow \rightarrow \exists r'. r \downarrow r'$ holds only classically.

We can now introduce a useful notion of *responsiveness*. A resumption r is responsive, if it keeps converging. It is defined coinductively with the help of the convergence predicate by

$$\frac{r \downarrow \text{ret } \sigma}{r \downarrow} [\text{resp-ret}] \quad \frac{r \downarrow \text{in } f \quad \forall v. (f v) \downarrow}{r \downarrow} [\text{resp-in}] \quad \frac{r \downarrow \text{out } v \quad r' \downarrow}{r \downarrow} [\text{resp-out}]$$

For instance, $\text{rep } n$, $\text{rep}' n$ and $\text{echo } \sigma$ are responsive, but \perp and echo' are not.

Classically, a resumption is responsive, if it can never evolve into a diverging resumption. Indeed, by augmenting the definition of responsiveness with a divergence option we obtain a classically tautological predicate, $r\Downarrow$, that we call *committedness*.

$$\frac{r \downarrow \text{ret } \sigma}{r \Downarrow} [\text{comm-ret}] \quad \frac{r \downarrow \text{in } f \quad \forall v. (f v) \Downarrow}{r \Downarrow} [\text{comm-in}] \quad \frac{r \downarrow \text{out } v \quad r' \Downarrow}{r \Downarrow} [\text{comm-out}] \quad \frac{r\uparrow}{r \Downarrow} [\text{comm-div}]$$

For a resumption r to be committed, it must be the case that it always either converges or diverges. So, classically, any resumption is committed.

Lemma 2.1 *Classically, for all r , $r\Downarrow$.*

Proof Specifically, we use an instance of excluded middle, $\forall r. (\exists r'. r \downarrow r') \vee \neg(\exists r'. r \downarrow r')$, which amounts to assuming that convergence is decidable. \square

Lemma 2.2 *Convergence, divergence, responsiveness and committedness are setoid predicates.*

3 Weak Bisimilarity

Two resumptions are weakly bisimilar, if they are bisimilar modulo collapsing finite sequences of delay steps between observable actions. It is conceivable that, in practice, weak bisimilarity is what is needed: one may well be interested only in observable behavior, disregarding finite delays. For instance, to guarantee correctness of a compiler optimization, we would want to prove that the optimization does not change the observable behavior of the source program, including termination and divergence behaviors, but the optimized code may perform fewer internal steps and thus be faster. We therefore formalize *termination-sensitive* weak bisimilarity, which distinguishes termination and silent divergence.

Technically, getting the definition of weak bisimilarity right is not straightforward, especially not in a constructive setting. It requires both induction and coinduction: we need to collapse a *finite* number of delay steps between observable actions possibly *infinitely*. Here we present two equivalent formulations (actually, we will also give a third one for classical reasoning, which is only equivalent to the first two classically). The first is closer to the formulations typically found in process calculi literature (except that, in process calculi, one usually works with termination-insensitive weak bisimilarity). The second nests induction into coinduction, exhibiting a useful technique for implementation in Coq. In our development,

we use both formulations and their equivalence result, freely choosing the one of the two that facilitates the proof.

The first one, noted $r \cong r_*$, uses coinduction atop the inductive definition of convergence and is defined by the rules

$$\frac{r \downarrow \text{ret } \sigma \quad r_* \downarrow \text{ret } \sigma}{r \cong r_*} \quad \frac{r \downarrow \text{in } f \quad r_* \downarrow \text{in } f_* \quad \forall v. f v \cong f_* v}{r \cong r_*} \quad \frac{r \downarrow \text{out } v \quad r_* \downarrow \text{out } v \quad r' \cong r'_*}{r \cong r_*} \quad \frac{r \cong r_*}{\delta r \cong \delta r_*}$$

so two resumptions are weakly bisimilar if they converge at the same action or can both perform an internal action, with weakly bisimilar residual resumptions. In particular, two terminating resumptions are derived to be weakly bisimilar by a single application of the first rule, whereas two silently diverging resumptions are weakly bisimilar by corecursive application of the fourth rule. For instance, we have $\text{rep } n \cong \text{rep}' n$ but $\text{echo } \sigma \cong \text{echo}'$ does not hold.

Lemma 3.1 *For any r, r' and r_* , if $r \downarrow r'$ and $r_* \uparrow$ then $\neg r \cong r_*$.*

As a corollary, we obtain that the silently diverging resumption \perp and resumptions that have terminated, $\text{ret } \sigma$, are not weakly bisimilar.

The second formulation, denoted $r \cong^\circ r_*$ nests induction into coinduction. We first define $\downarrow X \downarrow$ inductively in terms of X , for any relation (read: setoid relation) X , and then define \cong° coinductively in terms of $\downarrow \cong^\circ \downarrow$. For binary relations X, Y , $X \subseteq Y$ denotes $\forall x, x_*. x X x_* \rightarrow x Y x_*$.

$$\frac{}{\text{ret } \sigma \downarrow X \downarrow \text{ret } \sigma} \quad \frac{r X r_*}{\text{out } v \quad r \downarrow X \downarrow \text{out } v \quad r_*} \quad \frac{\forall v. f v X f_* v}{\text{in } f \downarrow X \downarrow \text{in } f_*} \quad \frac{r \downarrow X \downarrow r_*}{\delta r \downarrow X \downarrow r_*} \quad \frac{r \downarrow X \downarrow r_*}{r \downarrow X \downarrow \delta r_*} \\ \frac{X \subseteq \cong^\circ \quad r \downarrow X \downarrow r_*}{r \cong^\circ r_*} \quad \frac{r \cong^\circ r_*}{\delta r \cong^\circ \delta r_*}$$

Intuitively, $r \downarrow X \downarrow r_*$ means that r and r_* converge to resumptions related by X .

In the first rule of \cong° , we have used Mendler-style coinduction in order to enable Coq's syntactic guarded corecursion for \cong° . The natural (Park-style) rule to stipulate would have been:

$$\frac{r \downarrow \cong^\circ \downarrow r_*}{r \cong^\circ r_*}$$

Coq's guardedness condition for induction nested into coinduction is too weak to work with the Park-style rule: we cannot construct the corecursive functions (coinductive proofs) that we need. With our definition, the Park-style rule is derivable. We can also prove that $\downarrow X \downarrow$ is monotone in X , which allows us to recover the natural inversion principle for \cong° .

Induction and coinduction can be mixed in several ways. An inductive definition can be *mutual* with a coinductive definition, if the occurrence of one predicate in the definition of the other is contravariant.⁵ But this is not our situation. Instead, in our case, we have an inductive and a coinductive definition that use each other covariantly, but one is *nested* in the other. Specifically, we have the inductive definition nested in the coinductive definition⁶, since we want finite chunks of $\downarrow \cong^\circ \downarrow$ derivations to be weaved into an infinite \cong° derivation. The Agda developer community is currently exploring a novel approach to coinductive types (based on suspension types) [6, 7] where this form of mixing induction and coinduction is easily encoded while nesting the other way is problematic.

The two definitions of weak bisimilarity are equivalent.

⁵This means looking for a least X and greatest Y solving a system of equations $X = F(Y, X)$, $Y = G(X, Y)$, where F and G are contravariant in their first arguments and covariant in the second arguments.

⁶i.e., we have a definition of the form $\nu X. G(\mu Y. F(X, Y), X)$ with both F and G covariant in both arguments

Lemma 3.2 *For any r and r_* , $r \cong r_*$ iff $r \cong^\circ r_*$.*

Weak bisimilarity is a setoid predicate and an equivalence relation.

Lemma 3.3 *Weak bisimilarity is a setoid predicate: For any r, r', r_*, r'_* , if $r \approx r'$, $r \cong r_*$ and $r_* \approx r'_*$, then $r' \cong r'_*$. Weak bisimilarity is an equivalence.*

Proof Reflexivity and symmetry are straightforward to prove by coinduction. Below we sketch the proof for transitivity with the second formulation, $r \cong^\circ r_*$, to show Mendler-style coinduction working in our favour. For binary relations X, Y , let $X \circ Y$ denote their composition; namely, $x(X \circ Y)x'$ if there is x'' such that xXx'' and $x''Yx'$. We first prove, by *induction*, the transitivity for $\downarrow X \downarrow$, i.e., that, for any resumptions r_0, r_1, r_2 and setoid relations X, Y , if $r_0 \downarrow X \downarrow r_1$ and $r_1 \downarrow Y \downarrow r_2$, then $r_0 \downarrow (X \circ Y) \downarrow r_2$. The transitivity of \cong° states that, for any resumptions r_0, r_1 and r_2 , if $r_0 \cong^\circ r_1$ and $r_1 \cong^\circ r_2$, then $r_0 \cong^\circ r_2$. The proof of this is by *coinduction* and inversion on $r_0 \cong^\circ r_1$ and $r_1 \cong^\circ r_2$. We show the main case. Suppose we have $r_0 \cong^\circ r_1$ and $r_1 \cong^\circ r_2$, because $r_0 \downarrow X \downarrow r_1$ and $r_1 \downarrow Y \downarrow r_2$ for some X and Y such that $X \subseteq \cong^\circ$ and $Y \subseteq \cong^\circ$. By the transitivity of $\downarrow X \downarrow$ (which was proved by induction separately above), we obtain $r_0 \downarrow X \circ Y \downarrow r_2$. Using the coinduction hypothesis, we have $X \circ Y \subseteq \cong^\circ \circ \cong^\circ \subseteq \cong^\circ$, which closes the case. Notably, the invocation of the coinduction hypothesis here is properly guarded thanks to our use of Mendler's trick. \square

As one should expect, strongly bisimilar resumptions are weakly bisimilar.

Corollary 3.1 *For any r, r_* , $r \approx r_*$, then $r \cong r_*$.*

Proof Immediate from \cong being a reflexive setoid predicate. \square

Termination-sensitive bisimilarity has previously been considered by Kučera and Mayr [13] and Bohannon et al. [3] (but see also Bergstra et al. [1]). Their version is best suited for classical reasoning in the sense that terminating and silently diverging resumptions are distinguished by an upfront choice between convergence and divergence. This version of weak bisimilarity, denoted $r \cong_c r_*$, is defined coinductively by

$$\frac{r \downarrow \text{ret } \sigma \quad r_* \downarrow \text{ret } \sigma}{r \cong_c r_*} \quad \frac{r \downarrow \text{out } v \ r' \quad r_* \downarrow \text{out } v \ r'_* \quad r' \cong_c r'_*}{r \cong_c r_*} \quad \frac{r \downarrow \text{in } f \quad r_* \downarrow \text{in } f_* \quad \forall v. f \ v \cong_c f_* \ v}{r \cong_c r_*} \quad \frac{r \uparrow \quad r_* \uparrow}{r \cong_c r_*}$$

Only the fourth rule is different from the rules of \cong and refers directly to divergence.

The classical-style version of weak bisimilarity, \cong_c , is stronger than the constructive-style version, \cong . The converse is only true classically.

Lemma 3.4 *For any r and r_* , if $r \cong_c r_*$, then $r \cong r_*$. Classically, for any r and r_* , if $r \cong r_*$, then $r \cong_c r_*$.*

We insist on the use constructive-style weak bisimilarity, \cong , in the constructive setting, because the classical-style notion fails to enjoy some fundamental properties constructively.

Lemma 3.5 *Classical-style weak bisimilarity is a setoid predicate. Classically, it is also an equivalence weaker than strong bisimilarity.*

Proof We only prove that \cong_c is an equivalence. Reflexivity: We prove that for any r , $r \cong_c r$ by coinduction. Classically, we have $\forall r_0. (\exists r'_0. r_0 \downarrow r'_0) \vee r_0 \uparrow$. Should $r \uparrow$ hold, we immediately conclude $r \cong_c r$. Suppose there exists r' such that $r \downarrow r'$. Moreover suppose $r' = \text{in } f$ for some f . The coinduction hypothesis gives us that for any v , $f \ v \cong_c f \ v$, from which $r \cong_c r$ follows. The other cases, i.e., when $r' = \text{out } v \ r''$ for some v and r'' or $r' = \text{ret } \sigma$ for some σ , are similar. Symmetry: We prove constructively that for any r and r' , if $r \cong_c r'$ then $r' \cong_c r$ by coinduction and inversion on $r \cong_c r'$. Transitivity: We prove constructively that for any r, r' and r'' , if $r \cong_c r'$ and $r' \cong_c r''$ then $r \cong_c r''$ by coinduction and inversion on $r \cong_c r'$ and $r' \cong_c r''$. \square

Constructively, it is not possible to show classical-style weak bisimilarity reflexive and hence we cannot show any two strong bisimilar resumptions classical-style weakly bisimilar.

A simple example of a resumption r not classical-style weakly bisimilar to itself constructively is given by any search process that is classically total, but cannot be proved terminating constructively, since no bound on the search can be given. By definition, a resumption can only be classical-style weakly bisimilar to another if it terminates or diverges. Constructively, the resumption r is only nondiverging, we cannot show it terminating.

4 Big-Step Semantics

We now proceed to a first, basic (delayful) big-step operational semantics for our reactive While in terms of delayful resumptions. Evaluation $(s, \sigma) \Rightarrow r$, expressing that running a statement s from a state σ produces a resumption r , is defined coinductively by the rules in Figure 1. The rules for sequence and while implement the necessary sequencing with the help of extended evaluation $(s, r) \xRightarrow{*} r'$, expressing that running a statement s from the last state (if it exists) of an already accumulated resumption r results in a total resumption r' . Extended evaluation is also defined coinductively, as the coinductive prefix closure of evaluation.

Input and output statements produce corresponding resumptions that perform input or output actions and terminate thereafter. We consider assignments and testing of guards of if- and while-statements to constitute internal actions, observable as delays. This way we avoid introducing semantic anomalies, by making sure that any while-loop always progresses. But this choice also ensures that evaluation is total—as we should expect. Given that it is deterministic as well⁷, we can equivalently turn our relational big-step semantics into a functional one: the unique resumption for a given configuration (statement-state pair) is definable by corecursion.⁸ This semantics is a straightforward adaptation of the trace-based coinductive big-step semantics of non-interactive While from our previous work [17], where the details can be found and where we motivate all our design choices (e.g., why skip takes no time whereas the boolean guards do; we argue that our design is canonical).

Lemma 4.1 *Evaluation is a setoid predicate. It is total and deterministic up to bisimilarity.*

Let us look at some examples. We have $(\text{while true do skip}, \sigma) \Rightarrow \perp$ for any σ . I.e., while true do skip silently diverges. We also have $(\text{input } x; \text{while true do } (\text{output } x; x := x + 1), \sigma) \Rightarrow \text{in } (\lambda n. \text{up } n)$ where up is defined corecursively by $\text{up } n = \delta (\text{out } n (\delta (\text{up } (n + 1))))$. I.e., the statement counts up from the given input n . The two delays around every output action account for the internal actions of the assignment and testing of the boolean guard. An interactive adder takes two inputs and outputs their sum, and repeats this process, that is, we have $(\text{while true do } (\text{input } x; \text{input } y; \text{output } (x + y)), \sigma) \Rightarrow \text{sum}$ where sum is defined corecursively by $\text{sum} = \delta (\text{in } (\lambda m. \text{in } (\lambda n. \text{out } (m + n) \text{sum})))$.

Weak bisimilarity is useful for reasoning about soundness of program transformations, where we accept that transformations may change the timing of a resumption. For instance, we have $(\text{while true do } (z := x; \text{output } z), \sigma) \Rightarrow \text{rep } (\sigma x)$, where rep is defined corecursively by $\text{rep } n = \delta \delta (\text{out } n (\text{rep } n))$, and $(z := x; \text{while true do output } z, \sigma) \Rightarrow \delta (\text{rep}' (\sigma x))$, where rep' is defined corecursively by $\text{rep}' n = \delta (\text{out } n (\text{rep}' n))$, with $\text{rep } n \cong \text{rep}' n$. The latter resumption is faster than the former, but they are weakly

⁷Note that the external nondeterminism resulting from input actions is encapsulated in resumptions.

⁸This aspect makes our big-step operational semantics very close in spirit to denotational semantics, specifically, denotational semantics in terms of Kleisli categories, here, the Kleisli category of a resumptions monad.

$$\begin{array}{c}
\frac{}{(x := e, \sigma) \Rightarrow \delta (ret \sigma[x \mapsto \llbracket e \rrbracket \sigma])} \quad \frac{}{(skip, \sigma) \Rightarrow ret \sigma} \quad \frac{(s_0, \sigma) \Rightarrow r \quad (s_1, r) \xRightarrow{*} r'}{(s_0; s_1, \sigma) \Rightarrow r'} \\
\frac{e \models \sigma \quad (s_t, \delta (ret \sigma)) \xRightarrow{*} r}{(if \ e \ then \ s_t \ else \ s_f, \sigma) \Rightarrow r} \quad \frac{e \not\models \sigma \quad (s_f, \delta (ret \sigma)) \xRightarrow{*} r}{(if \ e \ then \ s_t \ else \ s_f, \sigma) \Rightarrow r} \\
\frac{e \models \sigma \quad (s_t, \delta (ret \sigma)) \xRightarrow{*} r \quad (while \ e \ do \ s_t, r) \xRightarrow{*} r'}{(while \ e \ do \ s_t, \sigma) \Rightarrow r'} \quad \frac{e \not\models \sigma}{(while \ e \ do \ s_t, \sigma) \Rightarrow \delta (ret \sigma)} \\
\frac{}{(input \ x, \sigma) \Rightarrow in (\lambda v. ret \sigma[x \mapsto v])} \quad \frac{}{(output \ e, \sigma) \Rightarrow out (\llbracket e \rrbracket \sigma) (ret \sigma)} \\
\frac{(s, \sigma) \Rightarrow r}{(s, ret \sigma) \xRightarrow{*} r} \quad \frac{\forall v. (s, f \ v) \xRightarrow{*} f' \ v}{(s, in \ f) \xRightarrow{*} in \ f'} \quad \frac{(s, r) \xRightarrow{*} r'}{(s, out \ v \ r) \xRightarrow{*} out \ v \ r'} \quad \frac{(s, r) \xRightarrow{*} r'}{(s, \delta \ r) \xRightarrow{*} \delta \ r'}
\end{array}$$

Figure 1: Basic (delayful) big-step semantics

bisimilar. In fact, we can prove while e do $(z := x; s)$ and $z := x; \text{while } e \text{ do } s$ to be weakly bisimilar whenever e is true of the initial state and s does not change x . Here, the latter statement is obtained from the former by loop-invariant code motion, a well-known compiler optimization; the optimization preserves the observable behaviour of the source statement, irrespective of its termination behaviour, which it must respect as well. We note that output 1 is not observationally equivalent to $(\text{while true do skip}); \text{output } 1$. More importantly, output 1 is not observationally equivalent to $\text{output } 1; (\text{while true do skip})$, since our weak bisimilarity is termination-sensitive. Of course, we can deal with more interesting program equivalences, such as the equivalence of $\text{mult} = \text{while true do } (\text{input } x; \text{input } y; z := 0; \text{while } x \neq 0 \text{ do } (z := z + y; x := x - 1); \text{output } z)$ and $\text{mult_opt} = \text{while true do } (\text{input } x; \text{input } y; \text{if } x \geq 0 \text{ then output } x * y \text{ else } (\text{while true do skip}))$, slow and fast interactive multipliers, which silently diverge when given a negative first operand.

5 Small-Step Semantics

In this section, we introduce an equivalent small-step semantics and define weak bisimilarity of configurations (statement-state pairs) in terms of it. We then prove two configurations to be weakly bisimilar if and only if their evaluations produce weakly bisimilar resumptions.

A configuration (s, σ) is a pair of a statement and state. *Labelled configurations* $c : lconf$ are defined by the sum⁹:

$$\frac{s : state}{ret \sigma : lconf} \quad \frac{s : stmt \quad g : Int \rightarrow state}{in \ s \ g : lconf} \quad \frac{v : Int \quad s : stmt \quad \sigma : state}{out \ v \ s \ \sigma : lconf} \quad \frac{s : stmt \quad \sigma : state}{\underline{\delta} \ s \ \sigma : lconf}$$

A terminality predicate/one-step reduction relation \rightarrow is defined in Figure 2 (top half). If $c = ret \ \sigma$, then the proposition $(s, \sigma) \rightarrow c$ means that the configuration (s, σ) has terminated at state σ . In other cases, it corresponds to a labelled transition: if $c = in \ s' \ g$, we take an input v and evolve to a configuration $(s', g \ v)$; if $c = out \ v \ s' \ \sigma'$, we output v and evolve to (s', σ') ; if $c = \underline{\delta} \ s' \ \sigma'$, the configuration (s, σ) evolves to a configuration (s', σ') in a delay step. We have chosen to label configurations rather than transitions so that labelled configurations become “trunks” of resumptions.

⁹The definition is non-recursive, but we pretend that it is inductive, as we also do in Coq.

$$\begin{array}{c}
\frac{}{(x := e, \sigma) \rightarrow \underline{\delta} \text{ skip } (\sigma[x \mapsto \llbracket e \rrbracket \sigma])} \quad \frac{}{(\text{skip}, \sigma) \rightarrow \underline{ret} \sigma} \\
\frac{(s_0, \sigma) \rightarrow \underline{ret} \sigma' \quad (s_1, \sigma') \rightarrow c}{(s_0; s_1, \sigma) \rightarrow c} \quad \frac{(s_0, \sigma) \rightarrow \underline{in} s'_0 f}{(s_0; s_1, \sigma) \rightarrow \underline{in} (s'_0; s_1) f} \quad \frac{(s_0, \sigma) \rightarrow \underline{out} v s'_0 \sigma'}{(s_0; s_1, \sigma) \rightarrow \underline{out} v (s'_0; s_1) \sigma'} \quad \frac{(s_0, \sigma) \rightarrow \underline{\delta} s'_0 \sigma'}{(s_0; s_1, \sigma) \rightarrow \underline{\delta} (s'_0; s_1) \sigma'} \\
\frac{\sigma \models e}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \rightarrow \underline{\delta} s_t \sigma} \quad \frac{\sigma \not\models e}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \rightarrow \underline{\delta} s_f \sigma} \\
\frac{\sigma \models e}{(\text{while } e \text{ do } s_t, \sigma) \rightarrow \underline{\delta} (s_t; \text{while } e \text{ do } s_t) \sigma} \quad \frac{\sigma \not\models e}{(\text{while } e \text{ do } s_t, \sigma) \rightarrow \underline{\delta} \text{ skip } \sigma} \\
\frac{}{(\text{input } x, \sigma) \rightarrow \underline{in} \text{ skip } (\lambda v. \sigma[x \mapsto v])} \quad \frac{}{(\text{output } e, \sigma) \rightarrow \underline{out} (\llbracket e \rrbracket \sigma) \text{ skip } \sigma} \\
\\
\frac{(s, \sigma) \rightarrow \underline{ret} \sigma'}{(s, \sigma) \rightsquigarrow \underline{ret} \sigma'} \quad \frac{(s, \sigma) \rightarrow \underline{\delta} s' \sigma' \quad (s', \sigma') \rightsquigarrow r}{(s, \sigma) \rightsquigarrow \delta r} \\
\frac{(s, \sigma) \rightarrow \underline{in} s' g \quad \forall v. (s', g v) \rightsquigarrow f v}{(s, \sigma) \rightsquigarrow \underline{in} f} \quad \frac{(s, \sigma) \rightarrow \underline{out} v s' \sigma' \quad (s', \sigma') \rightsquigarrow r}{(s, \sigma) \rightsquigarrow \underline{out} v r}
\end{array}$$

Figure 2: Small-step semantics

Weak bisimilarity of two configurations is defined in terms of terminality/one-step reduction. Again, convergence, $(s, \sigma) \downarrow c$, states that either (s, σ) terminates or performs an observable action in a finite number of steps. It is defined inductively by

$$\frac{(s, \sigma) \rightarrow \underline{ret} \sigma'}{(s, \sigma) \downarrow \underline{ret} \sigma'} \quad \frac{(s, \sigma) \rightarrow \underline{out} v s' \sigma'}{(s, \sigma) \downarrow \underline{out} v s' \sigma'} \quad \frac{(s, \sigma) \rightarrow \underline{in} s' g}{(s, \sigma) \downarrow \underline{in} s' g} \quad \frac{(s, \sigma) \rightarrow \underline{\delta} s' \sigma' \quad (s', \sigma') \downarrow c}{(s, \sigma) \downarrow c}$$

(We overload the same notations for resumptions and configurations without ambiguity.) Weak bisimilarity on configurations is defined coinductively by

$$\begin{array}{c}
\frac{(s, \sigma) \downarrow \underline{ret} \sigma' \quad (s_*, \sigma_*) \downarrow \underline{ret} \sigma'}{(s, \sigma) \cong (s_*, \sigma_*)} \\
\frac{(s, \sigma) \downarrow \underline{in} s' g \quad (s_*, \sigma_*) \downarrow \underline{in} s'_* g_* \quad \forall v. (s', g v) \cong (s'_*, g_* v)}{(s, \sigma) \cong (s_*, \sigma_*)} \\
\frac{(s, \sigma) \downarrow \underline{out} v (s', \sigma') \quad (s_*, \sigma_*) \downarrow \underline{out} v (s'_*, \sigma'_*) \quad (s', \sigma') \cong (s'_*, \sigma'_*)}{(s, \sigma) \cong (s_*, \sigma_*)} \\
\frac{(s, \sigma) \rightarrow \underline{\delta} s' \sigma' \quad (s_*, \sigma_*) \rightarrow \underline{\delta} s'_* \sigma'_* \quad (s', \sigma') \cong (s'_*, \sigma'_*)}{(s, \sigma) \cong (s_*, \sigma_*)}
\end{array}$$

Two configurations are weakly bisimilar if and only if their evaluations yield weakly bisimilar resumptions.

Lemma 5.1 *For any s, s_*, σ and σ_* , $(s, \sigma) \cong (s_*, \sigma_*)$ iff there exist r and r_* such that $(s, \sigma) \Rightarrow r$ and $(s_*, \sigma_*) \Rightarrow r_*$ and $r \cong r_*$.*

The evaluation relation of the small-step semantics is defined in Figure 2 (bottom half). It is the terminal many-step reduction relation, defined coinductively. The proposition $(s, \sigma) \rightsquigarrow r$ means that running s from the state σ produces the resumption r .

The big-step and small-step semantics are equivalent.

Proposition 5.1 *For any s, σ and r , $(s, \sigma) \Rightarrow r$ iff $(s, \sigma) \rightsquigarrow r$.*

6 Delay-Free Big-Step Semantics

So far we explicitly dealt with delay steps in a fully general and constructive manner. However, it is also possible to define big-step semantics in terms of resumptions without delay steps, by collapsing them on the fly, if they come in finite sequences. In this section, we define a delay-free semantics for configurations that lead to responsive resumptions.

We define *delay-free resumptions*, $r : res_r$, and their (strong) bisimilarity coinductively by

$$\frac{\sigma : state}{ret_r \sigma : res_r} \quad \frac{f : Int \rightarrow res_r}{in_r f : res_r} \quad \frac{v : Int \quad r : res_r}{out_r v r : res_r}$$

$$\frac{}{ret_r \sigma \approx ret_r \sigma} \quad \frac{\forall v. f v \approx f_* v}{in_r f \approx in_r f_*} \quad \frac{r \approx r_*}{out_r v r \approx out_r v r_*}$$

A responsive delayful resumption $r : res$ can be normalized into a delay-free resumption by collapsing the finite sequences of delay steps it has between observable actions. We define normalization, $norm : (r : res) \rightarrow r \Downarrow \rightarrow res_r$, and embedding of delay-free resumptions into delayful resumptions, $emb : res_r \rightarrow res$ by corecursion. In the definition of $norm$, we examine the proof of $r \Downarrow$, i.e., r 's responsiveness.

$$\begin{aligned} norm\ r\ (resp-ret\ \sigma\ _) &= ret_r\ \sigma & emb\ (ret_r\ \sigma) &= ret\ \sigma \\ norm\ r\ (resp-in\ f\ _k) &= in_r\ (\lambda v. norm\ (f\ v)\ (k\ v)) & emb\ (in_r\ f) &= in\ (\lambda v. emb\ (f\ v)) \\ norm\ r\ (resp-out\ v\ r'\ _h) &= out_r\ v\ (norm\ r'\ h) & emb\ (out_r\ v\ r) &= out\ v\ (emb\ r) \end{aligned}$$

A delayful resumption is weakly bisimilar to a delay-free one if and only if it is responsive and its normal form is strongly bisimilar to the same.

Lemma 6.1 *For any $r : res$ and $r_* : res_r$, $r \cong emb\ r_*$ iff $norm\ r\ h \approx r_*$ for some $h : r \Downarrow$.*

(The convergence proofs of a resumption are strong bisimilar, so h is unique up to that extent.)

Corollary 6.1 (i) *For any $r : res$, $h : r \Downarrow$, $r \cong emb\ (norm\ r\ h)$. (ii) *For any $r, h : r \Downarrow$ and $r_*, h_* : r_* \Downarrow$, $r \cong r_*$ iff $norm\ r\ h \approx norm\ r_*\ h_*$.**

In Figure 3, we define the delay-free big-step semantics for responsive programs. Here we have an inductive definition of a parameterized evaluation relation $\Rightarrow \downarrow(X)$ defined in terms of X , for any relation X , nested into a coinductive definition of an extended evaluation relation \Rightarrow^* , defined in terms of $\Rightarrow \downarrow(\Rightarrow^*)$. Finally, the actual evaluation relation \Rightarrow_r of interest is obtained by instantiating $\Rightarrow \downarrow$ at \Rightarrow^* . Since we collapse delay-steps on the fly, an assignment immediately terminates at the updated state. Likewise, testing the guard of a condition or a while-loop takes no time. The crucial rules are those for sequence and while-loop. If the first statement of a sequence or the body of a while-loop terminate silently, the second statement or the new iteration of the loop are run using the inductive evaluation. The coinductive extended evaluation is used only if the first statement or the body perform at least one input or output action.

This way, we make sure that only a finite number of delay steps may be collapsed between two observable actions, while allowing for diverging runs which perform input and output every now and then. Indeed, if we replaced the while-ret rule with

$$\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) ret_r \sigma' \quad (\text{while } e \text{ do } s_t, ret_r \sigma') \Rightarrow^* r'}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) r'}$$

we would obtain semantic anomalies. E.g., $(\text{while true do skip}, \sigma) \Rightarrow_r r$ would be derived for any $r : res_r$.

$$\begin{array}{c}
\frac{}{(x := e, \sigma) \Rightarrow \downarrow(X) \text{ret}_r (\sigma[x \mapsto \llbracket e \rrbracket \sigma])} \quad \frac{}{(\text{skip}, \sigma) \Rightarrow \downarrow(X) \text{ret}_r \sigma} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{ret}_r \sigma' \quad (s_1, \sigma') \Rightarrow \downarrow(X) r}{(s_0; s_1, \sigma) \Rightarrow \downarrow(X) r} \\
\frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{in}_r f \quad \forall v. (s_1, f v) X f' v}{(s_0; s_1, \sigma) \Rightarrow \downarrow(X) \text{in}_r f'} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{out}_r v r \quad (s_1, r) X r'}{(s_0; s_1, \sigma) \Rightarrow \downarrow(X) \text{out}_r v r'} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) r}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow(X) r} \quad \frac{e \not\models \sigma \quad (s_f, \sigma) \Rightarrow \downarrow(X) r}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow(X) r} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{ret}_r \sigma' \quad (\text{while } e \text{ do } s_t, \sigma') \Rightarrow \downarrow(X) r}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) r} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{in}_r f \quad \forall v. (\text{while } e \text{ do } s_t, f v) X f' v}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) \text{in}_r f'} \quad \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{out}_r v r \quad (\text{while } e \text{ do } s_t, r) X r'}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) \text{out}_r v r'} \\
\frac{e \not\models \sigma}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) \text{ret}_r \sigma} \\
\frac{}{(\text{input } x, \sigma) \Rightarrow \downarrow(X) \text{in}_r (\lambda v. \text{ret}_r \sigma[x \mapsto v])} \quad \frac{}{(\text{output } e, \sigma) \Rightarrow \downarrow(X) \text{out}_r (\llbracket e \rrbracket \sigma) (\text{ret}_r \sigma)} \\
\frac{X \subseteq \overset{*}{\Rightarrow} \quad (s, \sigma) \Rightarrow \downarrow(X) r}{(s, \text{ret}_r \sigma) \overset{*}{\Rightarrow} r} \quad \frac{\forall v. (s, f v) \overset{*}{\Rightarrow} f' v}{(s, \text{in}_r f) \overset{*}{\Rightarrow} \text{in}_r f'} \quad \frac{(s, r) \overset{*}{\Rightarrow} r'}{(s, \text{out}_r v r) \overset{*}{\Rightarrow} \text{out}_r v r'} \\
\frac{(s, \sigma) \Rightarrow \downarrow(\overset{*}{\Rightarrow}) r}{(s, \sigma) \Rightarrow_r r}
\end{array}$$

Figure 3: Delay-free big-step semantics

Coming back to the examples of the previous section, we have $(\text{input } x; \text{while true do } (\text{output } x; x := x + 1), \sigma) \Rightarrow_r \text{in } (\lambda n. \text{up}_r n)$ where up_r is defined corecursively by $\text{up}_r n = \text{out}_r n (\text{up}_r (n + 1))$. We also have $(\text{while true do } z := x; \text{output } z, \sigma) \Rightarrow_r \text{rep}_r (\sigma x)$ and $(z := x; \text{while true do } \text{output } z, \sigma) \Rightarrow \text{rep}_r (\sigma x)$ where rep_r is defined corecursively by $\text{rep}_r n = \text{out}_r n (\text{rep}_r n)$. Since the delay steps are collapsed on the fly in the delay-free semantics, the two statements produce the same, i.e., strongly bisimilar, (delay-free) resumptions. The delay-free semantics does not account for (i.e., does not assign a resumption to) non-responsive configurations, such as $\text{while true do skip}$ and the interactive multipliers from the previous section (since they diverge given a negative input for the first operand), with any initial state.

We state adequacy of the delay-free semantics by relating it to the delayful semantics of Section 4. Namely, for configurations leading to responsive resumptions they agree.

Proposition 6.1 (Soundness) *For any $s, \sigma, r : \text{res}_r$, if $(s, \sigma) \Rightarrow_r r$ then there exists $r' : \text{res}$ such that $(s, \sigma) \Rightarrow r'$ and $\text{emb } r \cong r'$.*

Proposition 6.2 (Completeness) *For any $s, \sigma, r : \text{res}$ and $h : r \Downarrow$, if $(s, \sigma) \Rightarrow r$, then $(s, \sigma) \Rightarrow_r \text{norm } r h$.*

The proofs are omitted due to the space limitation. They are nontrivial and the details can be found in the accompanying Coq development. Below we demonstrate the key proof technique on an example.

Consider the statement $\text{count} = \text{while true do } (\text{if } i > 0 \text{ then } i := i - 1 \text{ else } (\text{output } x; x := x + 1; i := x))$. It counts up from 0, so we should have $(\text{count}, \sigma) \Rightarrow_r \text{up}_r 0$ for an initial state σ that maps x and i to 0. We need coinduction since count performs outputs infinitely often; we also need induction, nested

into coinduction, since the loop silently iterates n times each time before outputting n . Note that the latency is finite but unbounded.

We cannot perform induction inside coinduction naïvely. That would be rejected by Coq’s syntactic guardedness checker, which is there to ensure productivity of coinduction. Mendler-style coinduction comes to rescue. Let $(s, r) R r'$ be a relation on pairs (s, r) of a statement and a resumption and resumptions r' , defined inductively by

$$\frac{}{(\text{count}, \text{ret}_r \sigma[x \mapsto n, i \mapsto n]) R \text{up}_r n} \quad \frac{(s, r) R r'}{(s, \text{out}_r v r) R \text{out}_r v r'} \quad \frac{(s, \sigma) \Rightarrow \downarrow(R) r}{(s, \text{ret}_r \sigma) R r}$$

The key fact is that R is stronger than \Rightarrow^* (Lemma 6.6 below).

We first prove that $\Rightarrow \downarrow$ is monotone by induction.

Lemma 6.2 *For any X, Y, s, σ and r such that $X \subseteq Y$, if $(s, \sigma) \Rightarrow \downarrow(X) r$, then $(s, \sigma) \Rightarrow \downarrow(Y) r$.*

The following two lemmata are proved by straightforward application of the rules in Figure 3.

Lemma 6.3 *For any n , (if $i > 0$ then $i := i - 1$ else (output $x; x := x + 1; i := x$), $\sigma[x \mapsto n, i \mapsto 0]) \Rightarrow \downarrow(R) \text{out}_r n (\text{ret}_r \sigma[x \mapsto n + 1, i \mapsto n + 1])$.*

Lemma 6.4 *For any n and m , (if $i > 0$ then $i := i - 1$ else (output $x; x := x + 1; i := x$), $\sigma[x \mapsto n, i \mapsto m + 1]) \Rightarrow \downarrow(R) \text{ret}_r \sigma[x \mapsto n, i \mapsto m]$.*

The next lemma is proved by induction on m , using the two lemmata just proved.

Lemma 6.5 *For any n and m , $(\text{count}, \sigma[x \mapsto n, i \mapsto m]) \Rightarrow \downarrow(R) \text{out}_r n (\text{up}_r (n + 1))$.*

Corollary 6.2 *For any n , $(\text{count}, \sigma[x \mapsto n, i \mapsto n]) \Rightarrow \downarrow(R) \text{up}_r n$.*

We can now prove that R is stronger than \Rightarrow^* by coinduction and inversion on $(s, r) R r'$. Here is the crux: corollary 6.2 together with the coinduction hypothesis gives $(\text{count}, \text{ret}_r \sigma[x \mapsto n, i \mapsto n]) \Rightarrow^* \text{up}_r n$, and the use of the coinduction hypothesis is properly guarded.

Lemma 6.6 *For any s, r and r' , if $(s, r) R r'$ then $(s, r) \Rightarrow^* r'$*

The main proposition follows from corollary 6.2, lemma 6.6 and the monotonicity of $\Rightarrow \downarrow$ (lemma 6.2).

Proposition 6.3 *For any n , $(\text{count}, \sigma[x \mapsto n, i \mapsto n]) \Rightarrow_r \text{up}_r n$.*

7 Classical-Style Big-Step Semantics

In Section 2, we augmented the definition of responsiveness with a divergence option to obtain a concept of committedness, which is a classically tautological predicate. Similarly, we can obtain a delay-free semantics for committed configurations from the delay-free semantics for responsive configurations of the previous section. To do so, we extend the definition of delay-free resumptions with a “black hole” constructor, \bullet , representing silent divergence, arriving at *classical-style resumptions*, and adjust the definition of (strong) bisimilarity:

$$\frac{\sigma : \text{state}}{\text{ret}_c \sigma : \text{res}_c} \quad \frac{f : \text{Int} \rightarrow \text{res}_c}{\text{in}_c f : \text{res}_c} \quad \frac{r : \text{res}_c}{\text{out}_c v r : \text{res}_c} \quad \frac{}{\bullet : \text{res}_c}$$

$$\frac{}{\text{ret}_c \sigma \approx \text{ret}_c \sigma} \quad \frac{\forall v. f v \approx f_* v}{\text{in}_c f \approx \text{in}_c f_*} \quad \frac{r \approx r_*}{\text{out}_c v r \approx \text{out}_c v r_*} \quad \frac{}{\bullet \approx \bullet}$$

Given a proof $h : r \Downarrow$ of committedness of a delayful resumption $r : res$, we can normalize r into a classical resumption by collapsing the finite delays between observable actions and sending silent divergence into the black hole.

$$\begin{array}{ll}
 \text{norm } r \text{ (comm-ret } \sigma \text{)} &= \text{ret}_c \sigma & \text{emb (ret}_c \sigma \text{)} &= \text{ret } \sigma \\
 \text{norm } r \text{ (comm-in } f \text{ } k \text{)} &= \text{in}_c (\lambda v. \text{norm } (f \text{ } v) (k \text{ } v)) & \text{emb (in}_c f \text{)} &= \text{in } (\lambda v. \text{emb } (f \text{ } v)) \\
 \text{norm } r \text{ (comm-out } v \text{ } r' \text{ } h \text{)} &= \text{out}_c v (\text{norm } r' \text{ } h) & \text{emb (out}_c v \text{ } r \text{)} &= \text{out } v (\text{emb } r) \\
 \text{norm } r \text{ (comm-div } \bullet \text{)} &= \bullet & \text{emb } \bullet &= \delta (\text{emb } \bullet)
 \end{array}$$

Again, a delayful resumption is weakly bisimilar to a classical-style one if and only if it is committed and its normal form is strongly bisimilar.

Lemma 7.1 *For any $r : res$ and $r_* : res_c$, $r \cong \text{emb } r_*$ iff $\text{norm } r \text{ } h \approx r_*$ for some $h : r \Downarrow$.*

In Figure 4, we define the classical-style semantics in terms of classical-style resumptions. We have an inductive parameterized evaluation relation $\Rightarrow \downarrow(X)$, defined in terms of X , for any relation X , for convergent runs; its inference rules are the same as those in the previous section. But we also have a coinductive parameterized evaluation $\Rightarrow \uparrow(X)$, again defined in terms of X , for any relation X , for silently diverging runs, so that $(s, \sigma) \Rightarrow \uparrow(\overset{*}{\Rightarrow})$ expresses that running a statement s from a state σ diverges without performing input or output. It uses the inductive evaluation in case the first statement of a sequence or the first iteration of the body of a while-loop silently terminates, but the whole sequence or while-loop silently diverges. Then we define coinductively an extended evaluation relation $\overset{*}{\Rightarrow}$, in terms of these two evaluation relations, nesting the latter into the former. Finally, we instantiate both $\Rightarrow \downarrow$ and $\Rightarrow \uparrow$ at $\overset{*}{\Rightarrow}$ to obtain the “real” evaluation relation \Rightarrow_c . Note that, to derive an evaluation proposition in this semantics, one has to decide upfront whether inductive or coinductive evaluation should be used—a decision that can be made classically, but not constructively.

The classical-style semantics is adequate wrt. the basic semantics of Section 4.

Proposition 7.1 (Soundness) *For any s, σ and $r : res_c$, if $(s, \sigma) \Rightarrow_c r$, then there exists $r' : res$ such that $(s, \sigma) \Rightarrow r'$ and $\text{emb } r \cong r'$.*

Proposition 7.2 (Completeness) *For any s, σ and $r : res$ and $h : r \Downarrow$, if $(s, \sigma) \Rightarrow r$, then $(s, \sigma) \Rightarrow_c \text{norm } r \text{ } h$.*

Corollary 7.1 *Classically, for any s, σ and $r : res$, if $(s, \sigma) \Rightarrow r$, then there exists $r' : res_c$ such that $(s, \sigma) \Rightarrow_c r'$ and $r \cong \text{emb } r'$.*

The classical-style semantics is more expressive than responsive semantics, since it offers the option of “detected” divergence. In particular we have $(\text{while true do skip}, \sigma) \Rightarrow_c \bullet$ and our interactive multipliers are assigned a classical-style resumption *mult* defined corecursively by $\text{mult} = \text{in}_c (\lambda m. \text{in}_c (\lambda n. \text{if } m \geq 0 \text{ then } \text{out}_c (m * n) \text{ } \text{mult} \text{ else } \bullet))$; i.e., we have $(\text{mult}, \sigma) \Rightarrow_c \text{mult}$ and $(\text{mult_opt}, \sigma) \Rightarrow_c \text{mult}$.

8 Related Work

Formalized semantics are an important ingredient in the trusted computing base of certified compilers. Proof assistants, like Coq, are a good tool for such formalization projects, as both the object semantics of interest and its metatheory can be developed in the same framework. For introductions, see [2].

To account for nontermination or silent divergence properly in a big-step semantics is nontrivial already for languages without interactive I/O. Leroy and Grall [14] introduced two big-step semantics

$$\begin{array}{c}
\frac{}{(x := e, \sigma) \Rightarrow \downarrow(X) \text{ret}_c (\sigma[x \mapsto \llbracket e \rrbracket \sigma])} \quad \frac{}{(\text{skip}, \sigma) \Rightarrow \downarrow(X) \text{ret}_c \sigma} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{ret}_c \sigma' \quad (s_1, \sigma') \Rightarrow \downarrow(X) r}{(s_0; s_1, \sigma) \Rightarrow \downarrow(X) r} \\
\frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{in}_c f \quad \forall v. (s_1, f v) X f' v}{(s_0; s_1, \sigma) \Rightarrow \downarrow(X) \text{in}_c f'} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{out}_c v r \quad (s_1, r) X r'}{(s_0; s_1, \sigma) \Rightarrow \downarrow(X) \text{out}_c v r'} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) r}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow(X) r} \quad \frac{e \not\models \sigma \quad (s_f, \sigma) \Rightarrow \downarrow(X) r}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow(X) r} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{ret}_c \sigma' \quad (\text{while } e \text{ do } s_t, \sigma') \Rightarrow \downarrow(X) r}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) r} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{in}_c f \quad \forall v. (\text{while } e \text{ do } s_t, f v) X f' v}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) \text{in}_c f'} \quad \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{out}_c v r \quad (\text{while } e \text{ do } s_t, r) X r'}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) \text{out}_c v r'} \\
\frac{e \not\models \sigma}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow(X) \text{ret}_c \sigma} \\
\frac{}{(\text{input } x, \sigma) \Rightarrow \downarrow(X) \text{in}_c (\lambda v. \text{ret}_c \sigma[x \mapsto v])} \quad \frac{}{(\text{output } e, \sigma) \Rightarrow \downarrow(X) \text{out}_c (\llbracket e \rrbracket \sigma) (\text{ret}_c \sigma)} \\
\frac{(s_0, \sigma) \Rightarrow \uparrow(X)}{(s_0; s_1, \sigma) \Rightarrow \uparrow(X)} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow(X) \text{ret}_c \sigma' \quad (s_1, \sigma') \Rightarrow \uparrow(X)}{(s_0; s_1, \sigma) \Rightarrow \uparrow(X)} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \uparrow(X)}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \uparrow(X)} \quad \frac{e \not\models \sigma \quad (s_f, \sigma) \Rightarrow \uparrow(X)}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \uparrow(X)} \\
\frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \uparrow(X)}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \uparrow(X)} \quad \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow(X) \text{ret}_c \sigma' \quad (\text{while } e \text{ do } s_t, \sigma') \Rightarrow \uparrow(X)}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \uparrow(X)} \\
\frac{X \subseteq \overset{*}{\Rightarrow} (s, \sigma) \Rightarrow \downarrow(X) r}{(s, \text{ret}_c \sigma) \overset{*}{\Rightarrow} r} \quad \frac{X \subseteq \overset{*}{\Rightarrow} (s, \sigma) \Rightarrow \uparrow(X)}{(s, \text{ret}_c \sigma) \overset{*}{\Rightarrow} \bullet} \quad \frac{\forall v. (s, f v) \overset{*}{\Rightarrow} f' v}{(s, \text{in}_c f) \overset{*}{\Rightarrow} \text{in}_c f'} \quad \frac{(s, r) \overset{*}{\Rightarrow} r'}{(s, \text{out}_c v r) \overset{*}{\Rightarrow} \text{out}_c v r'} \quad \frac{}{(s, \bullet) \overset{*}{\Rightarrow} \bullet} \\
\frac{(s, \sigma) \Rightarrow \downarrow(\overset{*}{\Rightarrow}) r}{(s, \sigma) \Rightarrow_c r} \quad \frac{(s, \sigma) \Rightarrow \uparrow(\overset{*}{\Rightarrow})}{(s, \sigma) \Rightarrow_c \bullet}
\end{array}$$

Figure 4: Classical-style big-step semantics

for lambda-calculus. One is classical in spirit, with two evaluation relations, inductive and coinductive, for terminating and diverging runs, and relies on decidability between termination and divergence. The other, with a single coinductive evaluation relation, is essentially suited for constructive reasoning, but contains a semantic anomaly (a function can continue reducing after the argument diverges), which results from its ability to collapse an infinite sequence of internal actions (contraction steps).

In our work [17] on While with nontermination, we developed a trace-based coinductive big-step semantics where traces were non-empty colists of intermediate states, agreeing with the very standard coinductive small-step trace-based semantics. This semantics relied on traces being a monad; a central component in the definition was an extended evaluation relation, corresponding to the Kleisli extension of evaluation. Capretta [4] studied constructive denotational semantics of nontermination as the Kleisli semantics for the delayed state monad, corresponding to hiding the intermediate states in the trace monad as internal actions and quotienting by termination-sensitive weak bisimilarity. Rutten [21] carried out a similar project in classical set theory where the quotient is the state space extended with an extra element

for nontermination.

Operational semantics of interactive programs is most often described in the small-step style where it amounts to a labelled transition system. Especially, this is the dominating approach in process calculi. Big-step semantics is closer to denotational semantics. In this field, resumption-based descriptions go back to Plotkin [20], Gunter et al. [10] and Cenciarelli and Moggi [5]. Resumptions are a monad and resumptions-based denotational semantics is a Kleisli semantics. Our big-step semantics are directly inspired by this approach, except that we work in a constructive setting and must take extra care to avoid the need to invoke classical principles where they are dispensable.

We are not aware of many other works on constructive semantics of interactive I/O. But similar in its spirit to ours is the work of Hancock et al. [11] on stream processors and the stream functions that these induce by “eating”. Stream processors are like our delay-free resumptions, except that the authors emphasize parallel composition of stream processors (one processor’s output becomes another processor’s input) and, for this to be well-defined, a stream processor must not terminate and may only do a finite number of input actions consecutively. Hancock et al. [8] also characterize realizable stream functions. In a precursor work, Hancock and Setzer [12] studied a model of interaction where a client sends a server commands and expects responses.

Weak bisimilarity tends to be defined termination-insensitively, identifying termination and divergence. In particular, this is also the approach of CCS [16]. Termination-sensitive weak bisimilarity has been considered by Bergstra, Klop and Olderog [1], Kučera and Mayr [13] and Bohannon et al. [3], but only in what we call the classical-style version, relying on decisions between convergence and divergence. (The weak bisimilarity of Capretta [4] is termination-sensitive and tailored for constructive reasoning, but restricted to behaviours without I/O. Weak bisimilarity also motivated the study of Danielsson and Altenkirch [6] on mixed induction-coinduction.)

Mixed inductive-coinductive definitions in the form of induction nested into coinduction ($\nu X. \mu Y. F(X, Y)$ or, more generally, $\nu X. G(\mu Y. F(X, Y), X)$) seem to be quite fundamental in applications (e.g., the stream processors of Hancock et al., our delay-free semantics). Danielsson and Altenkirch [6, 7] argue for making this mix the basic form of inductive-coinductive definitions in the dependently-typed programming language Agda. Unfortunately, nestings the other way around (definitions $\mu X. \nu Y. F(X, Y)$) seem to become difficult or impossible to code. With our approach, coinduction nested into induction is handled symmetrically to induction nested into coinduction [19].

Mendler-style (co)recursion originates from Mendler [15]. It uses that a monotone (co)inductive definition is equivalent to a positive one, via a syntactic left (right) Kan extension along identity (instead of $\mu X. F X$ one works with $\mu X. \exists Y. (Y \rightarrow X) \rightarrow F Y$). We exploited this fact to enable Coq’s guarded corecursion for a coinductive definition with a nested inductive definition, at the price of impredicativity.

We have previously developed and formalized a Hoare logic for the trace-based semantics of While with nontermination [18]. A similar enterprise should be possible for resumptions, weak bisimilarity and While with interactive I/O.

9 Conclusion

We have developed a constructive treatment of resumption-based big-step semantics of While with interactive I/O. We have devised constructive-style definitions of important concepts on resumptions such as termination-sensitive weak bisimilarity and responsiveness, and devised two variations of delay-free big-step semantics for programs that produce responsive and committed resumptions, respectively. Responsiveness is for interactive computation what termination is for noninteractive computation. And

likewise, committedness compares to a decided domain of definedness. Indeed, all three variations of big-step semantics for While with interactive I/O have counterparts in big-step semantics for noninteractive While (see Appendix). Mathematically, we find it reassuring that observations made for a more simpler noninteractive While naturally scale to a more involved language with interactive I/O. The central ideas are a concept of termination-sensitive weak bisimilarity tailored for constructive reasoning and the organization of evaluation in the delay-free semantics as an induction nested into coinduction.

Technically, we have carried out an advanced exercise in programming and reasoning with mixed induction and coinduction, which we have also formalized in Coq. The challenges in this exercise were both mathematical and tool-related (Coq-specific). We deem that the mathematical part was more interesting and important. The main new aspect in comparison to our earlier development of coinductive trace-based big-step semantics for noninteractive While was the need to deal with definitions of predicates that nest induction into coinduction—a relatively unexplored area in type theory. In Coq, we formalized them by parameterizing the inductive definition and converting the coinductive definition into Mendler-like format. Apparently, this technique is novel for the Coq community.

As future work, we would like to scale our development to concurrency.

Acknowledgments We thank Andreas Lochbihler, Nils Anders Danielsson and Thorsten Altenkirch for discussions.

This research was supported by the Estonian Centre of Excellence in Computer Science, EXCS, funded by the European Regional Development Fund, and the Estonian Science Foundation grant no. 6940.

References

- [1] J. Bergstra, J. Klop & E.-R. Olderog (1987): *Failures without chaos: A new process semantics for fair abstraction*. In: M. Wirsing (ed.) *Proc. of 3rd IFIP TC2/WG2.2 Working Conf. on Formal Description of Programming Concepts (Ebberup, Aug. 1986)*. North Holland, Amsterdam, pp. 77–101.
- [2] Y. Bertot (2007): *A survey of programming language semantics styles*. Coq development, <http://www-sop.inria.fr/marelle/Yves.Bertot/proofs.html>
- [3] A. Bohannon, B. C. Pierce, V. Sjöberg, S. Weirich & S. Zdancewic (2009): *Reactive noninterference*. In: *Proc. of 2009 ACM Conf. on Computer and Communications Security, CCS 2009 (Chicago, IL, Nov. 2009)*, ACM Press, New York, pp. 79–90.
- [4] V. Capretta (2005): *General recursion via coinductive types*. *Logical Methods in Computer Science* 1(2), article 1.
- [5] P. Cenciarelli & E. Moggi (1993): *A syntactic approach to modularity in denotational semantics*. In: *Proc. of 5th Biennial Meeting on Category Theory and Computer Science, CTCS '93 (Amsterdam, Sept. 1993)* Tech. report, CWI, Amsterdam.
- [6] N. A. Danielsson & T. Altenkirch (2009): *Mixing induction and coinduction*. Draft, <http://www.cs.nott.ac.uk/~nad/publications/>.
- [7] N. A. Danielsson & T. Altenkirch (2010): Subtyping, declaratively: an exercise in mixed induction and coinduction. In C. Bolduc, J. Desharnais & B. Ktari (Eds.): *Proc. of 10th Int. Conf. on Mathematics of Program Construction, MPC 2010 (Québec City, July 2010)*, *Lect. Notes in Comput. Sci.* 6120. Springer, Berlin, pp. 100–118.
- [8] M. Ghani, P. Hancock & D. Pattinson (2009): *Continuous functions on final coalgebras*. In: S. Abramsky, M. Mislove & C. Palamidessi (eds.) *Proc. of 25th Conf. on Mathematical Foundations of Programming Semantics, MFPS-25 (Oxford, Apr. 2009)*. *Electr. Notes in Theor. Comput. Sci.* 249. Elsevier, Amsterdam, pp. 3–18.

- [9] E. Giménez (1995): Codifying guarded definitions with recursive schemes. In P. Dybjer, B. Nordström & J. M. Smith (Eds.): *Selected Papers from Int. Wksh. on Types for Proofs and Programs, TYPES '94* (Båstad, June 1994), *Lect. Notes in Comput. Sci.* 996. Springer, Berlin, pp. 39–59
- [10] C. A. Gunter, P. D. Mosses & D. S. Scott (1989): *Semantic Domains and Denotational Semantics*.
- [11] P. Hancock, D. Pattinson & N. Ghani (2009): *Representations of stream processors using nested fixed points*. *Logical Methods in Computer Science* 5(3), article 9.
- [12] P. Hancock & A. Setzer, A. (2000): *Interactive programs in dependent type theory*. In: P. Clote & H. Schwichtenberg (eds.): *Proc. of 14th Int. Wksh. on Computer Science Logic, CSL 2000* (Fischbachau, Aug. 2000). *Lect. Notes in Comput. Sci.* 1862. Springer, Berlin, pp. 317–331.
- [13] A. Kučera & R. Mayr (2002): *Weak bisimilarity between finite-state systems and BPA or normed BPP is decidable in polynomial time*. *Theor. Comput. Sci.* 270(1–2), pp. 677–700.
- [14] X. Leroy & H. Grall (2009): *Coinductive big-step operational semantics*. *Inform. and Comput.* 207(2), pp. 285–305.
- [15] N. P. Mendler (1991): *Inductive types and type constraints in the second-order lambda calculus*. *Ann. of Pure and Appl. Logic* 51(1–2), pp. 159–172.
- [16] R. Milner (1989): *Communication and Concurrency*. Prentice Hall, New York.
- [17] K. Nakata & T. Uustalu (2009): *Trace-based coinductive operational semantics for While: big-step and small-step, relational and functional styles*. In: S. Berghofer, T. Nipkow, C. Urban & M. Wenzel (eds.) *Proc. of 22nd Int. Conf. on Theorem Proving in Higher-Order Logics, TPHOLs 2009* (Munich, Aug. 2009). *Lect. Notes in Comput. Sci.* 5674. Springer, Berlin, pp. 375–390.
- [18] K. Nakata & T. Uustalu (2010): *A Hoare logic for the coinductive trace-based big-step semantics of While*. In: A. D. Gordon (ed.) *Proc. of 19th Europ. Symp. on Programming, ESOP 2010* (Paphos, March 2010). *Lect. Notes in Comput. Sci.* 6012. Springer, Berlin, pp. 488–506.
- [19] K. Nakata & T. Uustalu (2010): *Mixed induction-coinduction at work for Coq (abstract)*. Abstract of talk presented at 2nd Coq Workshop (Edinburgh, July 2010), with accompanying slides and Coq development.
- [20] G. D. Plotkin (1983): *Domains (“Pisa Notes”)*. Unpublished notes.
- [21] J. Rutten (1999): *A note on coinduction and weak bisimilarity for While programs*. *Theor. Inform. and Appl.* 33(4-5), pp. 393–400.

A Resumptions, Weak Bisimilarity, Delayful, Delay-Free and Classical-Style Big-step Semantics for While

The notions of resumptions and weak bisimilarity and the evaluation relations in the three big-step semantics shown of the main text are fairly involved, because of the amount of detail. Therefore, we also spell out what they specialize (or degenerate) to in the case of ordinary non-interactive While, to better highlight the phenomena that arise even in the absence of interaction.

A.1 Resumptions, Bisimilarity, Weak Bisimilarity

Delayful resumptions, with their strong bisimilarity, specialize to delayed states $r : res$ à la Capretta [4] defined coinductively.

$$\frac{\sigma : state}{ret \sigma : res} \quad \frac{r : res}{\delta r : res} \quad \frac{}{ret \sigma \approx ret \sigma} \quad \frac{r \approx r_*}{\delta r \approx \delta r_*}$$

Convergence and (silent) divergence are defined inductively resp. coinductively; convergence reduces to termination at a final state.

$$\frac{}{ret\ \sigma \downarrow ret\ \sigma} \quad \frac{r \downarrow r'}{\delta\ r \downarrow r'} \quad \frac{r \uparrow}{\delta\ r \uparrow}$$

Responsiveness reduces to termination. Committedness becomes decidability between and termination and divergence. Committedness is tautologically true only classically.

Weak bisimilarity is defined in terms of convergence coinductively exactly as Capretta [4] did.

$$\frac{r \downarrow ret\ \sigma \quad r_* \downarrow ret\ \sigma}{r \cong r_*} \quad \frac{r \cong r_*}{\delta\ r \cong \delta\ r_*}$$

Any terminating delayed state can be normalized into a state. Any decided delayed state can be normalized into a choice between a state or a special divergence token.

A.2 Delayful Semantics

In the delayful big-step semantics, evaluation and extended evaluation are defined mutually coinductively as follows.

$$\begin{array}{c} \frac{}{(x := e, \sigma) \Rightarrow \delta (ret\ \sigma[x \mapsto \llbracket e \rrbracket \sigma])} \quad \frac{}{(\text{skip}, \sigma) \Rightarrow ret\ \sigma} \quad \frac{(s_0, \sigma) \Rightarrow r \quad (s_1, r) \xRightarrow{*} r'}{(s_0; s_1, \sigma) \Rightarrow r'} \\[10pt] \frac{e \models \sigma \quad (s_t, \delta (ret\ \sigma)) \xRightarrow{*} r}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow r} \quad \frac{e \not\models \sigma \quad (s_f, \delta (ret\ \sigma)) \xRightarrow{*} r}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow r} \\[10pt] \frac{e \models \sigma \quad (s_t, \delta (ret\ \sigma)) \xRightarrow{*} r \quad (\text{while } e \text{ do } s_t, r) \xRightarrow{*} r'}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow r'} \quad \frac{e \not\models \sigma}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \delta (ret\ \sigma)} \\[10pt] \frac{(s, \sigma) \Rightarrow r}{(s, ret\ \sigma) \xRightarrow{*} r} \quad \frac{(s, r) \xRightarrow{*} r'}{(s, \delta\ r) \xRightarrow{*} \delta\ r'} \end{array}$$

We have previously [17] conducted a thorough study of a variation of this semantics (with intermediate states instead of delays), explaining the design considerations in great detail. We have also [18] developed a Hoare logic for this semantics.

A.3 Delay-Free Semantics

Delay-free resumptions are the same as states.

In the delay-free semantics, there is one inductive evaluation relation for terminating configurations. There is no need for a separate extended evaluation relation (which would coincide with evaluation anyhow, since resumptions and states are the same thing) and no need to parameterize the evaluation relation.

$$\begin{array}{c} \frac{}{(x := e, \sigma) \Rightarrow \downarrow \sigma[x \mapsto \llbracket e \rrbracket \sigma]} \quad \frac{}{(\text{skip}, \sigma) \Rightarrow \downarrow \sigma} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow \sigma' \quad (s_1, \sigma') \Rightarrow \downarrow \sigma''}{(s_0; s_1, \sigma) \Rightarrow \downarrow \sigma''} \\[10pt] \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow \sigma'}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow \sigma'} \quad \frac{e \not\models \sigma \quad (s_f, \sigma) \Rightarrow \downarrow \sigma'}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow \sigma'} \\[10pt] \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow \sigma' \quad (\text{while } e \text{ do } s_t, \sigma') \Rightarrow \downarrow \sigma''}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow \sigma''} \quad \frac{e \not\models \sigma}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow \sigma} \end{array}$$

The delay-free semantics agrees with the delayful semantics for terminating delayed states.

It is the textbook big-step semantics of While, which accounts for terminating configurations and assigns no evaluation result to diverging configurations.

A.4 Classical-Style Semantics

A classical-style resumption is a state or the special token \bullet for divergence.

$$\frac{\sigma : \text{state}}{\text{ret}_c \sigma : \text{res}_c} \quad \frac{}{\bullet : \text{res}_c}$$

The classical-style semantics has an inductively defined terminating evaluation relation (defined exactly as that of the delay-free semantics) and a coinductively defined diverging evaluation relation. The latter depends on the former, but not the other way around. There is no need for an extended evaluation relation.

$$\begin{array}{c} \frac{}{(x := e, \sigma) \Rightarrow \downarrow \sigma[x \mapsto \llbracket e \rrbracket \sigma]} \quad \frac{}{(\text{skip}, \sigma) \Rightarrow \downarrow \sigma} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow \sigma' \quad (s_1, \sigma') \Rightarrow \downarrow \sigma''}{(s_0; s_1, \sigma) \Rightarrow \downarrow \sigma''} \\[10pt] \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow \sigma'}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow \sigma'} \quad \frac{e \not\models \sigma \quad (s_f, \sigma) \Rightarrow \downarrow \sigma'}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \downarrow \sigma'} \\[10pt] \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow \sigma' \quad (\text{while } e \text{ do } s_t, \sigma') \Rightarrow \downarrow \sigma''}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow \sigma''} \quad \frac{e \not\models \sigma}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \downarrow \sigma} \\[10pt] \frac{(s_0, \sigma) \Rightarrow \uparrow}{(s_0; s_1, \sigma) \Rightarrow \uparrow} \quad \frac{(s_0, \sigma) \Rightarrow \downarrow \sigma' \quad (s_1, \sigma') \Rightarrow \uparrow}{(s_0; s_1, \sigma) \Rightarrow \uparrow} \quad \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \uparrow}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \uparrow} \quad \frac{e \not\models \sigma \quad (s_f, \sigma) \Rightarrow \uparrow}{(\text{if } e \text{ then } s_t \text{ else } s_f, \sigma) \Rightarrow \uparrow} \\[10pt] \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \uparrow}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \uparrow} \quad \frac{e \models \sigma \quad (s_t, \sigma) \Rightarrow \downarrow \sigma' \quad (\text{while } e \text{ do } s_t, \sigma') \Rightarrow \uparrow}{(\text{while } e \text{ do } s_t, \sigma) \Rightarrow \uparrow} \\[10pt] \frac{(s, \sigma) \Rightarrow \downarrow \text{ret}_c \sigma'}{(s, \sigma) \Rightarrow_c \sigma'} \quad \frac{(s, \sigma) \Rightarrow \uparrow}{(s, \sigma) \Rightarrow_c \bullet} \end{array}$$

The classical-style semantics agrees with the delayful semantics for decided delayed states (classically, any delayed state is decided).

A semantics in this spirit (with separate convergent and divergent evaluation relations) was proposed for untyped lambda calculus by Leroy and Grall [14].

The delayful semantics (together with the identification of weakly bisimilar delayed states) and the classical-style semantics have the same purposes, but the delayful semantics is better behaved from the constructive point-of-view. As a practical consequence, it has the advantage that the evaluation relation can be turned into a function (highly desirable, if one wants to be able to directly execute the big-step semantics). This is not possible with the classical-style semantics, as one would have to be able to decide whether a configuration terminates before actually running it.